

Porting a C program to CONCEPTUAL

```

/*
 * Copyright (C) 2002-2004 the Network-Based Computing Laboratory
 * (NBCL), The Ohio State University.
 */

#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>

#define MESSAGE_ALIGNMENT 64
#define MAX_MSG_SIZE (1<<22)
#define MYBUFSIZE (MAX_MSG_SIZE + MESSAGE_ALIGNMENT)

char s_buf_original[MYBUFSIZE];
char r_buf_original[MYBUFSIZE];

int skip = 1000;
int loop = 10000;
int skip_large = 10;
int loop_large = 100;
int large_message_size = 8192;

int main(int argc, char *argv[])
{
    int myid, numprocs, i;
    int size;
    MPI_Status reqstat;
    char *s_buf, *r_buf;
    int align_size;

    double t_start = 0.0, t_end = 0.0;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    align_size = MESSAGE_ALIGNMENT;

    s_buf =
        (char *) (((unsigned long) s_buf_original + (align_size - 1)) /
                  align_size * align_size);
    r_buf =
        (char *) (((unsigned long) r_buf_original + (align_size - 1)) /
                  align_size * align_size);

    if (myid == 0) {
        fprintf(stdout, "# OSU MPI Latency Test (Version 2.0)\n");
        fprintf(stdout, "# Size\tLatency (us)\n");
    }

    for (size = 0; size <= MAX_MSG_SIZE;
         size = (size ? size * 2 : size + 1)) {

        /* touch the data */
        for (i = 0; i < size; i++) {
            s_buf[i] = 'a';
            r_buf[i] = 'b';
        }

        if (size > large_message_size) {
            loop = loop_large;
            skip = skip_large;
        }

        MPI_Barrier(MPI_COMM_WORLD);

        if (myid == 0) {
            for (i = 0; i < loop + skip; i++) {
                if (i == skip)
                    t_start = MPI_Wtime();
                MPI_Send(s_buf, size, MPI_CHAR, 1, 1, MPI_COMM_WORLD);
                MPI_Recv(r_buf, size, MPI_CHAR, 1, 1, MPI_COMM_WORLD,
                         &reqstat);
            }
            t_end = MPI_Wtime();

        } else if (myid == 1) {
            for (i = 0; i < loop + skip; i++) {
                MPI_Recv(r_buf, size, MPI_CHAR, 0, 1, MPI_COMM_WORLD,
                         &reqstat);
                MPI_Send(s_buf, size, MPI_CHAR, 0, 1, MPI_COMM_WORLD);
            }
        }

        if (myid == 0) {
            double latency;
            latency = (t_end - t_start) * 1.0e6 / (2.0 * loop);
            fprintf(stdout, "%d\t%0.2f\n", size, latency);
        }
    }

    MPI_Finalize();
    return 0;
}

```

C (73 lines)

```

# coNCePTuaL version of the OSU MPI latency test
# Code written by Scott Pakin <pakin@lanl.gov>

# Ensure that future changes to the coNCePTuaL language don't break this
# program.
Require language version "0.5.3b".

# For the user's convenience we convert the C version's hardwired
# constants to command-line arguments.
message_alignment is "Message buffer alignment (bytes)" and comes from
    "--align" or "-a" with default 64.
max_msg_size is "Maximum message size (bytes)" and comes from "--max-size"
    or "-m" with default 4M.
skip is "Number of small-message warmup iterations" and comes from "--skip" or
    "-k" with default 1000.
loop is "Number of small-message iterations" and comes from "--loop" or "-l" with
    default 10000.
skip_large is "Number of large-message warmup iterations" and comes from
    "--skip-large" or "-j" with default 10.
loop_large is "Number of large-message iterations" and comes from "--loop-large" or
    "-i" with default 100.
large_message_size is "Large message size (bytes)" and comes from "--large-size"
    or "-s" with default 8K.

# Here's an extra bonus, courtesy of coNCePTuaL.
Assert that "The latency test requires at least two nodes" with num_tasks>=2.

Let s_buf be 0 and r_buf be 1 while {
    for each size in {0, 1, 2, 4, ..., max_msg_size} {
        all tasks touch all message buffers then
        let loop be loop_large if size > large_message_size otherwise loop
        and skip be skip_large if size > large_message_size otherwise skip while {
            all tasks synchronize then
            task 0 resets its counters then
            for loop repetitions plus skip warmup repetitions {
                task 0 sends a size-byte message_alignment-byte-aligned message
                from buffer s_buf to task 1 who receives it into buffer r_buf then
                task 1 sends a size-byte message_alignment-byte-aligned message
                from buffer s_buf to task 0 who receives it into buffer r_buf
            } then
            task 0 logs size as "Size" and elapsed_usecs/total_msgs as "Latency (us)"
        }
    }
}

```

CONCEPTUAL (24 lines)

Advantages of the CONCEPTUAL version

- ★ Shorter
- ★ More readable
- ★ Portable to other messaging layers without source-code changes
- ★ Logs not only performance data but also a wealth of information about the experimental setup—timer type & quality, compilers used, environment variables, complete source code, and more
- ★ Replaces hardwired constants with command-line options (and even supports --help)
- ★ Fails gracefully when run with too few processors